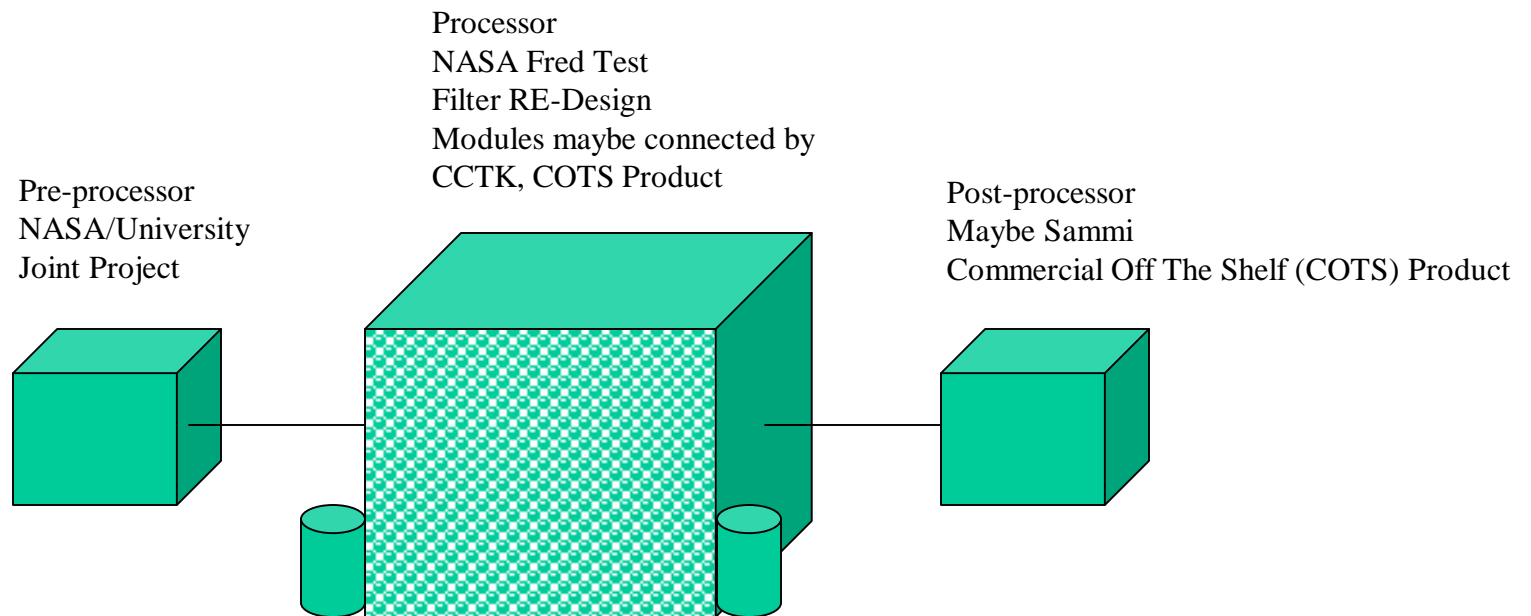


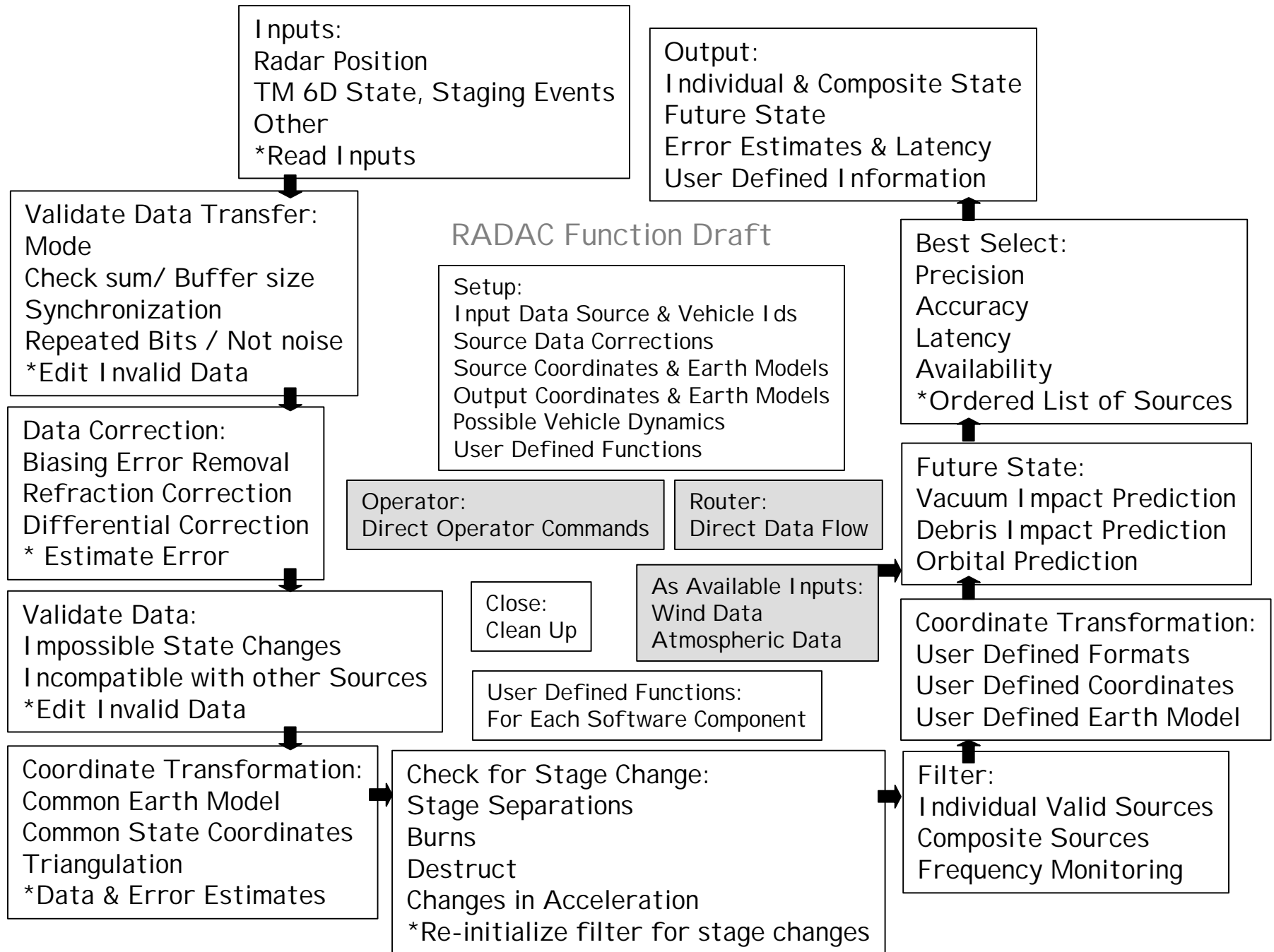
The following pages contain suggestions for the new RADAC. They include ideas expressed by our re-design team members and by other Wallops employees. Suggestions include a list of standard functions for the RADAC processor, the idea of identifying data sets with a tracker-vehicle-user tag, possible setup file organization, possible data structures, function input, function output, additional control functions, additional programs on the RADAC system.

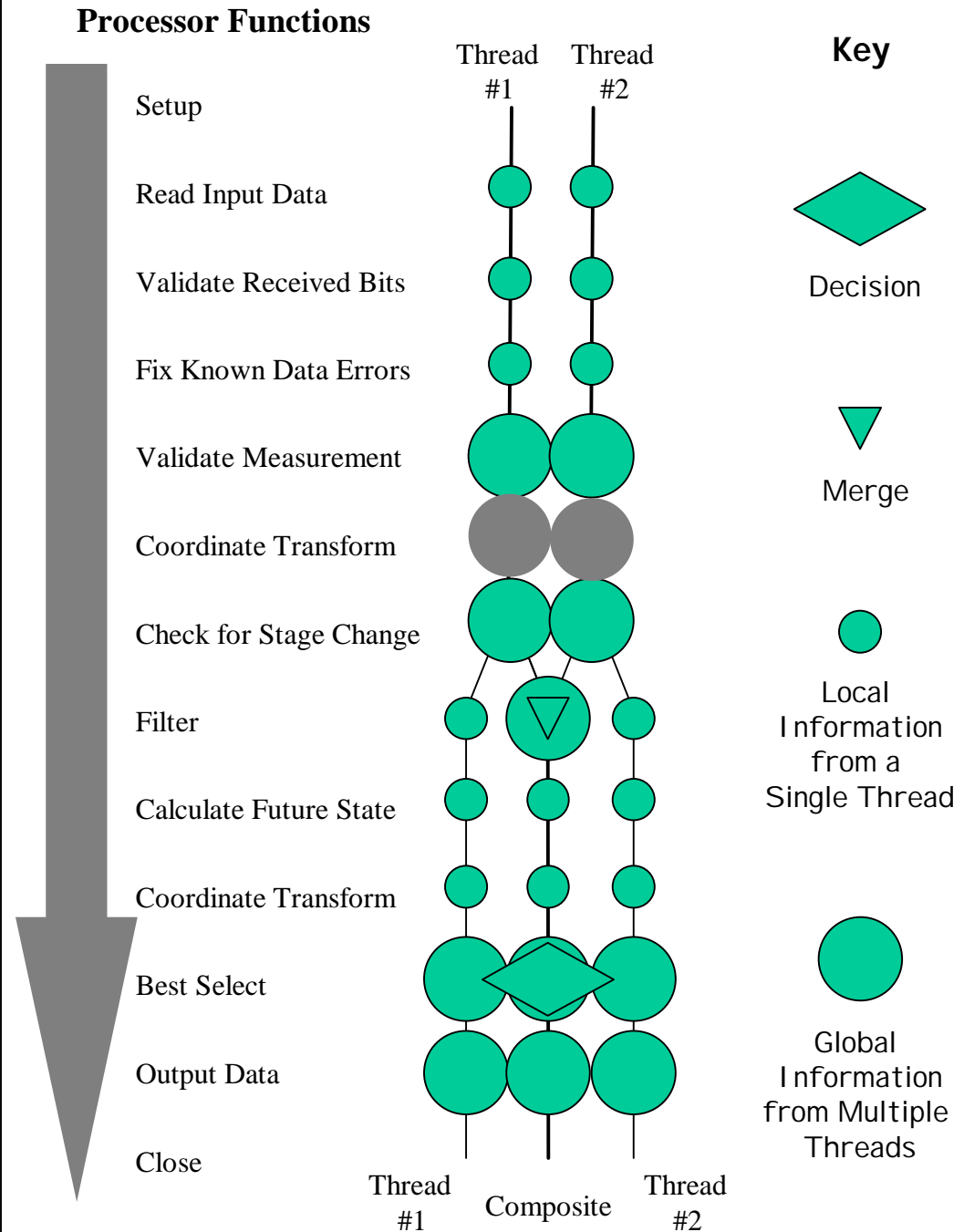
I'm trying out the presented organization in a Microsoft Development Studio Visual C++ program called FRED (Filter RE-Design). The primary purpose for creating FRED is to try out different filter classes.

In creating this presentation, I realized that I don't have a good idea for how to combine different data sets into a composite set. Also, the setup files are huge. Suggestions would be appreciated.

RADAC Re-design



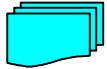




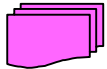
Each RADAC Application Preflight Configuration Includes:



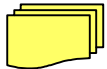
Only ONE Mission File



MULTIPLE Tracker Files for Each Device



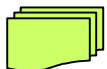
MULTIPLE Vehicle Files for Each Tracker



MULTIPLE User Files for Each Vehicle



ONE Standard Functions File for Each Thread



ONE Extra Functions File for Each Thread

Each Data Thread has a four part identification,
representing of the Thread's path through the Mission File Tree.

Example: MDDF Line 1, Radar 11, Pegasus Rocket, Range Safety

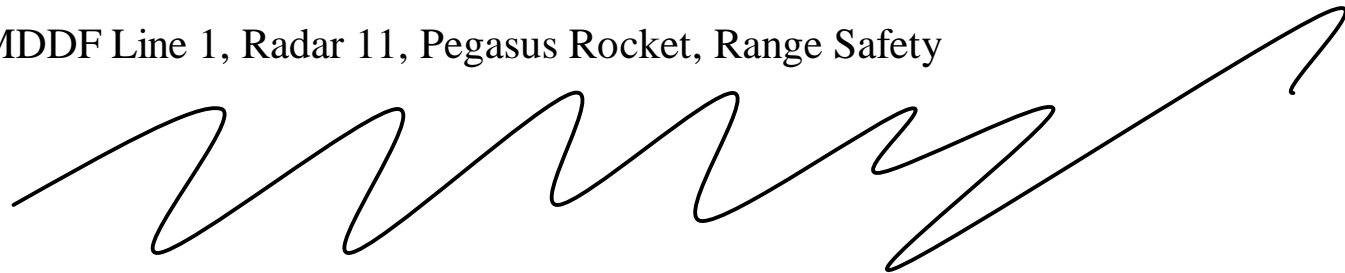
Thread ID:

Input Device #

Tracker #

Vehicle #

User #



```
# of Input Devices
Input Device Name #1
  # of Tracker Files for Input Device #1
    Tracker File Name #1 for Input Device #1
      # of Vehicle Files for Tracker File #1 for Input Device #1
        Vehicle File Name #1 for Tracker File #1 for Input Device #1
          # of User Files for Vehicle File #1 for Tracker File #1 for Input Device #1
            User File Name #1 for Vehicle File #1 for Tracker File #1 for Input Device #1
              Standard Files File Name for User #1, Vehicle #1, Tracker #1, Device #1
              Extra Files File Name for User #1, Vehicle #1, Tracker #1, Device #1
            User File Name #2 for Vehicle File #1 for Tracker File #1 for Input Device #1
              Standard Files File Name for User #2, Vehicle #1, Tracker #1, Device #1
              Extra Files File Name for User #2, Vehicle #1, Tracker #1, Device #1
          ...
        Vehicle File Name #2 for Tracker File #1 for Input Device #1
          # of User Files for Vehicle File #2 for Tracker File #1 for Input Device #1
            User File Name #1 for Vehicle File #2 for Tracker File #1 for Input Device #1
              Standard Files File Name for User #1, Vehicle #2, Tracker #1, Device #1
              Extra Files File Name for User #1, Vehicle #2, Tracker #1, Device #1
            User File Name #2 for Vehicle File #2 for Tracker File #1 for Input Device #1
              Standard Files File Name for User #2, Vehicle #2, Tracker #1, Device #1
              Extra Files File Name for User #2, Vehicle #2, Tracker #1, Device #1
          ...
        ...
      Tracker File Name #2 for Input Device #1
        # of Vehicle Files for Tracker File #2 for Input Device #1
          Vehicle File Name #1 for Tracker File #2 for Input Device #1
            # of User Files for Vehicle File #1 for Tracker File #2 for Input Device #1
              User File Name #1 for Vehicle File #1 for Tracker File #2 for Input Device #1
                Standard Files File Name for User #1, Vehicle #1, Tracker #2, Device #1
                Extra Files File Name for User #1, Vehicle #1, Tracker #2, Device #1
              User File Name #2 for Vehicle File #1 for Tracker File #2 for Input Device #1
                Standard Files File Name for User #2, Vehicle #1, Tracker #2, Device #1
                Extra Files File Name for User #2, Vehicle #1, Tracker #2, Device #1
            ...
          Vehicle File Name #2 for Tracker File #2 for Input Device #1
            # of User Files for Vehicle File #2 for Tracker File #2 for Input Device #1
              User File Name #1 for Vehicle File #2 for Tracker File #2 for Input Device #1
                Standard Files File Name for User #1, Vehicle #2, Tracker #2, Device #1
                Extra Files File Name for User #1, Vehicle #2, Tracker #2, Device #1
              User File Name #2 for Vehicle File #2 for Tracker File #2 for Input Device #1
                Standard Files File Name for User #2, Vehicle #2, Tracker #2, Device #1
                Extra Files File Name for User #2, Vehicle #2, Tracker #2, Device #1
            ...
          ...
        ...
      ...
    ...
  ...

```

A Mission File is a Tree of Other File Names.

Content of Five Input File Types (Excluding the Mission File):

Tracker(s)	Vehicle(s)	User(s)	Standard Functions for Thread(s)	Extra Functions for Thread(s)
			Setup	End of Setup
Input Format	-----	-----	Read Input Data	End of Input
Possible Verifications	-----	-----	Validate Received Bits	End of Validate Bits
Known Errors	-----	-----	Fix Known Data Errors	End of Error Correct
Tracker Limits	Vehicle Limits	-----	Validate Measurement	End of Validate Value
Tracker Coordinates	Filter Coordinates	-----	Coordinate Transform	End of Transform
-----	Possible Checks	-----	Check for Stage Change	End of Stage Check
Error Estimate	Filter Model	-----	Filter	End of Filter
-----	Filter Coordinates	Output Coordinates	Coordinate Transform	End of Transform
-----	-----	Required Output	State Calculations	End of State Calculate
-----	-----	Optimum Qualities	Best Select	End of Best Select
-----	-----	Output Format	Output Data	End of Output
			Close	End of Close

Data Structures for Archival on the Pre-processor Storage Unit

- **SCHEME Structure**

- Stored each time the set up function runs
- Contains the MISSION substructure
- Contains THREAD substructures

Under each THREAD are substructures for each file, TRACKER, VEHICLE, USER, STANDARD_FUNCTIONS, EXTRA_FUNCTIONS

- **SOURCE Structure**

- Stored each time the read data function runs
- Contains all information received by the input device

Data Structures for Archival on the Post-processor Storage Unit

- **HUMAN Structure**
 - Stored each time an operator interacts with RADAC
 - Contains the states of all possible operator inputs
- **RADAC Structure**
 - Stored each time the output data function runs
 - Contains substructures to hold data (excluding SCHEME, SOURCE, and HUMAN data) transferred between functions
- **EXTRA Structure**
 - Stored each time the output data function runs
 - Contains arrays of user defined values

Setup Function

- Initiation: Human specifies a mission file
- Input: mission file name
- Function: Read the specified mission file and its associated files, to fill the SCHEME structure. Archive the SCHEME structure. Initialize specified functions and RADAC variables.
- Output: SCHEME, RADAC, notification of completion to the operator

Input Function

- Initiation: Input device senses data arrival.
- Input: external data,
SCHEME.TRACKER.INPUT_FORMAT
- Function: Read the external data, to fill the SOURCE structure. Archive the SOURCE structure. Send SOURCE to the data transfer validation function.
- Output: SOURCE

Validate Transfer Function

- Initiation: Call from router after the input function ends
- Input: SOURCE,
SCHEME.TRACKER.VERIFICATIONS
- Function: Validate data transfer. Set RADAC structure's state variable. Decide if data is worth further processing.
- Output: RADAC.data_state_i

Data Correction Function

- Initiation: Call from router after validate transfer function completion with good RADAC.data_state_i
- Input: SOURCE, SCHEME.TRACKER.KNOWN_ERRORS
- Function: Correct the SOURCE data for know errors. Store the result in a RADAC substructure, called REFINED.
- Output: RADAC.REFINED

Validate Values Function

- Initiation: Call from router after data correction function completion
- Input: RADAC.REFINED,
SCHEME.TRACKER.LIMITS,
SCHEME.VEHICLE.LIMITS
- Function: Validate data values based on tracker and vehicle limitations. Revise RADAC structure's state variable. Decide if data is worth further processing.
- Output: RADAC.data_state_i

Transform Tracker Function

- Initiation: Call from router after validate values function completion with good RADAC.data_state_i
- Input: RADAC.REFINED,
SCHEME.TRACKER.COORDINATES,
SCHEME.VEHICLE.COORDINATES
- Function: Transform from the tracker coordinates to the filter coordinates.
- Output: RADAC.MEASUREMENT

Staging Check Function

- Initiation: Call from router after transform tracker function completion
- Input: RADAC.MEASUREMENT,
SCHEME.VEHICLE.STAGING_CHECKS
- Function: Look for duplicated indications that a stage change is occurring. Upon sensing a stage change, set the filter's filter_state_i variable to re-initialize.
- Output: RADAC.filter_state_i

Filter Function

- Initiation: Call from router after staging check function completion
- Input: RADAC.MEASUREMENT,
RADAC.filter_state_i,
SCHEME.TRACKER.ERROR_ESTIMATES,
SCHEME.VEHICLE.FILTER_MODEL
- Function: Filter the measurement data.
- Output: RADAC.FILTERED

Transform Filter Function

- Initiation: Call from router after filter function completion
- Input: RADAC.FILTERED,
SCHEME.VEHICLE.COORDINATES,
SCHEME.USER.COORDINATES
- Function: Transform from the filter coordinates to the output coordinates.
- Output: RADAC.PROCESSED.PRESENT

Future State Function

- Initiation: Call from router after transform filter function completion
- Input: RADAC.FILTERED,
SCHEME.USER.REQUIRED_STATE
- Function: Performed impact predictions or orbital predictions on the filtered data, to meet user prediction requirements.
- Output: RADAC.PROCESSED.FUTURE

Best Select Function

- Initiation: Call from router after future state function completion
- Input: RADAC.PROCESSED.PRESENT, RADAC.PROCESSED.FUTURE, SCHEME.USER.OPTIMIZE
- Function: Order all threads for the defined user, based on user optimization criteria.
- Output: RADAC.PROCESSED.OPTIMIZE

Output Function

- Initiation: Call from router after best select function completion
- Input: RADAC
- Function: Pack each user's RADAC.PROCESSED thread structures. Distribute the data to the users. Archive the entire RADAC structure.
- Output: RADAC.PROCESSED data packet leaving the RADAC, archived RADAC data

Close Function

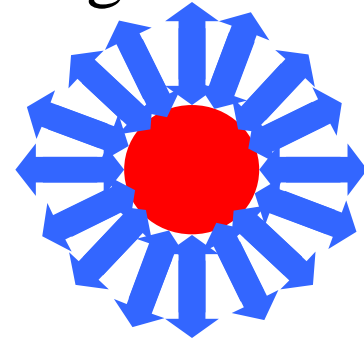
- Initiation: Human initiates close command
- Input: SCHEME
- Function: Gracefully complete the application.
End all functions. End archival. Close all files.
Clear all memory.
- Output: notification of completion to the operator

Extra Function(s)

- Initiation: notification from the router
- Input: SCHEME, SOURCE, HUMAN, RADAC, EXTRA
- Function: User defined
- Output: SCHEME, SOURCE, HUMAN, RADAC, EXTRA

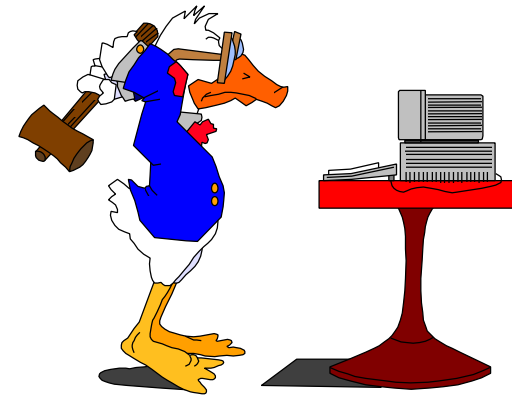
Router Function

- Initiation: notification of completion of any other function
- Input: SCHEME, completed function return ID
- Function: Call functions in the sequence specified by the thread's SCHEME, while the RADAC.data_state_i variable indicates good data. Combines and splits threads.
- Output: call to next function



Operator Function

- Initiation: operator input to the RADAC
- Input: HUMAN, SCHEME, RADAC
- Function: Update the appropriate data based on the state change in the HUMAN structure.
- Output: SCHEME, RADAC



New Available Function

- Initiation: notification that a replacement file is available
- Input: HUMAN
- Function: Obtain replacement item information from the HUMAN.NEW_AVAILABLE structure. Check the new data for errors. Replace the old with the new. Verify replacement.
- Output: notification that the replacement was or was not completed successfully



Stored Structures

SCHEME

- MISSION
 - mission_file_name_c[maximum_string_size]
- THREAD[# of threads]
 - thread_ID_c[maximum string size]
- TRACKER
 - INPUT_FORMAT
 - format_ID_c[maximum_string_size]
 - VERIFICATIONS
 - number_of_verifications_i
 - CHECK[# of verifications]
 - verification_ID_c[maximum_string_size]
 - values_i[# of verification integers]
 - values_d[# of verification doubles]
 - values_s[maximum string size][# of verification strings]
 - KNOWN_ERRORS
 - number_of_errors_i
 - ERROR[# of known errors]
 - error_ID_c[maximum_string_size]
 - values_i[# of error integers]
 - values_d[# of error doubles]
 - values_s[maximum string size][# of error strings]
 - LIMITS
 - number_of_limits_i
 - TEST[# of limits]
 - limit_ID_c[maximum_string_size]
 - values_i[# of limit integers]
 - values_d[# of limit doubles]
 - values_s[maximum string size][# of limit strings]

Stored Structures

- COORDINATES

- number_of_coordinates_i
- FRAME[# of coordinates]
 - coordinate_ID_c[maximum_string_size]
 - values_i[# of coordinate frame integers]
 - values_d[# of coordinate frame doubles, ex. Reference Location]
 - values_s[maximum string size][# of coordinate frame strings]

- ERROR_ESTIMATES

- number_of_estimates_i
- QUALITY[# of estimates]
 - estimate_ID_c[maximum_string_size]
 - values_i[# of estimate integers]
 - values_d[# of estimate doubles]
 - values_s[maximum string size][# of estimate strings]

- VEHICLE

- LIMITS

- number_of_limits_i
 - TEST[# of limits]
 - limit_ID_c[maximum_string_size]
 - values_i[# of limit integers]
 - values_d[# of limit doubles]
 - values_s[maximum string size][# of limit strings]

- COORDINATES

- number_of_coordinates_i
- FRAME[# of coordinates]
 - coordinate_ID_c[maximum_string_size]
 - values_i[# of coordinate frame integers]
 - values_d[# of coordinate frame doubles, ex. Reference Location]
 - values_s[maximum string size][# of coordinate frame strings]

Stored Structures

- STAGING_CHECKS
 - number_of_checks_i
 - CHECK[# of checks]
 - check_ID_c[maximum_string_size]
 - values_i[# of check integers]
 - values_d[# of check doubles]
 - values_s[maximum string size][# of check strings]
- FILTER_MODEL
 - model_ID_c[maximum_string_size]
- USER
 - COORDINATES
 - number_of_coordinates_i
 - FRAME[# of coordinates]
 - coordinate_ID_c[maximum_string_size]
 - values_i[# of coordinate frame integers]
 - values_d[# of coordinate frame doubles, ex. Reference Location]
 - values_s[maximum string size][# of coordinate frame strings]
 - REQUIRED_STATE
 - number_of_requirements_i
 - requirement_ID_c[maximum string size][# of requirements]
 - OPTIMIZE
 - number_of_qualities_i
 - QUALITY[# of coordinates]
 - quality_ID_c[maximum string size]
 - quality-weight_d
- STANDARD_FUNCTIONS
 - PROCESSOR[# of standard functions, in standard processing order]
 - standard_function_ID_c[maximum string size]
 - function_name_c[maximum string size]
- EXTRA_FUNCTIONS
 - USER_DEFINED[# of user defined functions]
 - standard_function_ID_to_follow_c[maximum string size]
 - function_name_c[maximum string size]

Stored Structures

HUMAN

- NEW_AVAILABLE
 - affected_item_ID_c[maximum string size]
 - replacement_name_c[maximum string size]
 - replacement_location_c[maximum string size]
- INITIATE_SETUP
 - mission_file_name_c[maximum string size]
- INITIATE_CLOSE
 - mission_file_name_c[maximum string size]
- REMOVE_EXTRA_FUNCTION
 - function_name_c[maximum string size]
- ADD_EXTRA_FUNCTION
 - standard_function_ID_to_follow_c[maximum_string_size]
 - function_name_c[maximum string size]
- CHANGE_STANDARD_FUNCTION
 - standard_function_ID_c[maximum_string_size]
 - function_name_c[maximum string size]
- FORCE_DATA_STATE
 - new_data_state_i
 - hold_state_flag_I (FALSE = one time, TRUE = until close)
- FORCE_FILTER_STATE
 - new_filter_state_i
 - hold_state_flag_I
- THREAD_CONTROL
 - thread_ID_c[maximum string size] (specific or determined from tracker, vehicle, & user types)
 - tracker_name_c[maximum string size] (specific or “all” of the specified vehicle & user types)
 - vehicle_name_c[maximum string size] (specific or “all” of the specified tracker & user types)
 - user_name_c[maximum string size] (specific or “all” of the specified tracker & vehicle types)
 - include_flag_i (FALSE = take out of system, TRUE = put in system, default is TRUE)

* The HUMAN structure should be stored with a time tag, to allow re-play capability.

* The HUMAN structure should include substructures for every control/command the operator can alter/enter.

Stored Structures

RADAC

- data_state_i
- filter_state_i
- REFINED
 - state_vector_d[maximum model degree][maximum coordinate axes]
 - time_tag_d
- MEASUREMENT
 - state_vector_d[maximum model degree][maximum coordinates axes]
 - time_tag_d
- FILTERED
 - state_vector_d[maximum model degree][maximum coordinates axes]
 - time_tag_d
- PROCESSED
 - PRESENT
 - state_vector_d[maximum model degree][maximum coordinates axes]
 - time_tag_d
 - FUTURE
 - state_vector_d[maximum model degree][maximum coordinates axes]
 - time_tag_d
 - OPTIMIZE
 - quality_i

■ The standard product for the user

Stored Structures

EXTRA

- values_i[# of user defined integers]
- values_d[# of user defined doubles]
- values_s[maximum string size][# of user defined strings]

■ An additional product for the user

SOURCE

- values_i[# of source integers]
- values_d[# of source doubles]
- values_s[maximum string size][# of source strings]

* To understand the SOURCE, you have to have format information from the SCHEME. Both should be stored on the same external unit.

* Dynamic memory allocation may save significant space.

Additional Programs

Setup Program

The SCHEME structure is large. It is filled using the setup files. However, setup does not have to be labor intensive. TRACKER, VEHICLE, USER, and even MISSION files can be defined once and re-used. The creator of these files, should use a setup program that lists necessary information and possible selections. The setup program can order the information in an ASCII text file.

Each file type should have it's own directory. Each file should have a standard, descriptive name.

The setup program should be separate from the RADAC program, but it should reside on the same system. Running the setup program from a remote terminal should be possible.

Test Program

Insuring that all portions of the code are tested is a must for accepting system changes. The more the system can change, the more the system needs testing. A test program should accelerate this process. It should search for code inconsistencies. It should contain archived setups with archived results, and it should be capable of comparing new results to the old results.

Replay Program

This program should read the user products (RADAC.PROCESSED and EXTRAS) from the archive. It should re-send this data, through the output function, at a rate matching the change in the data time tags.

Archive Extract Program

This program should read operator defined data from archives and store the data in a file. The file format one of a limited number of choices, selected by the operator. Some post processing routines may be available. Data reduction can be performed on a separate system, using the extracted file.